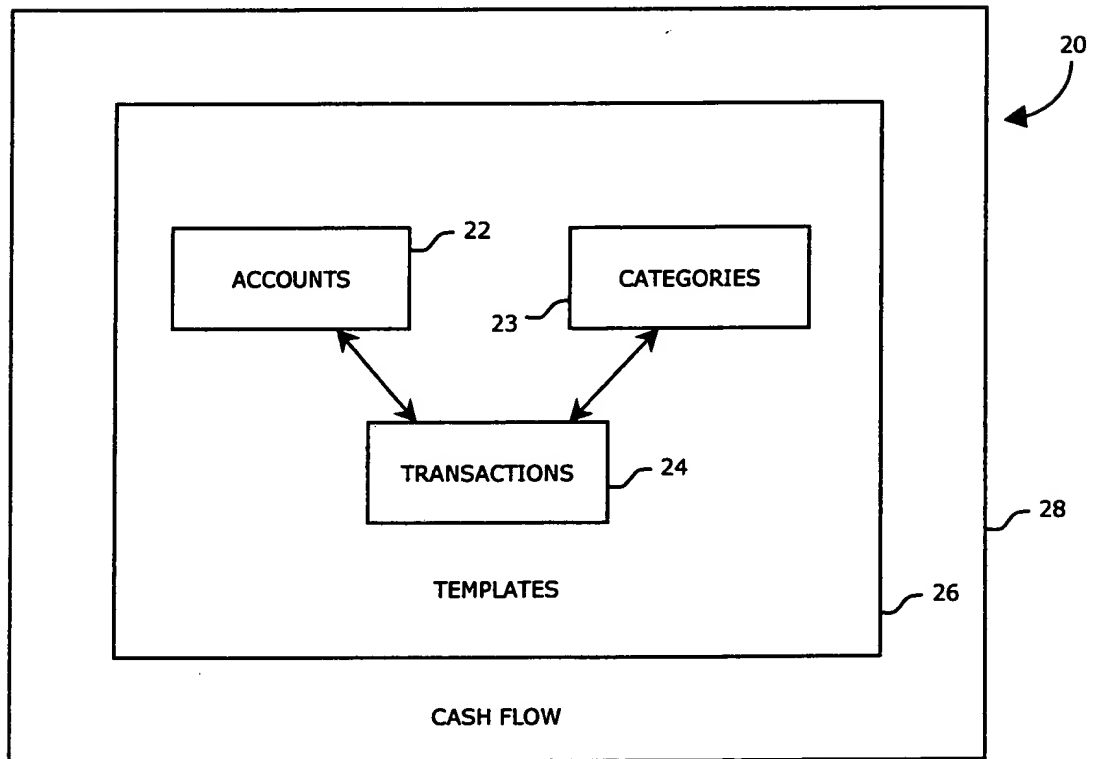


**FIG. 1**



**FIG. 2**

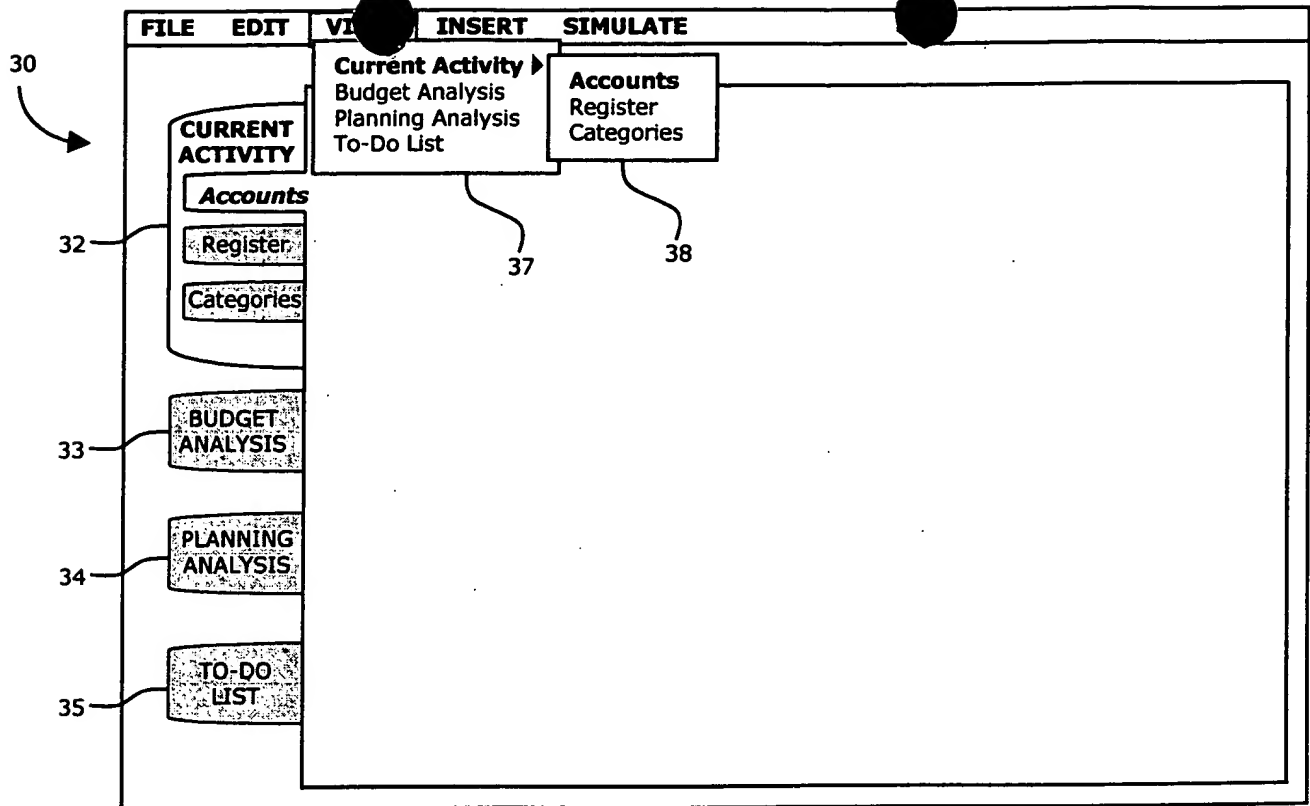


FIG. 3

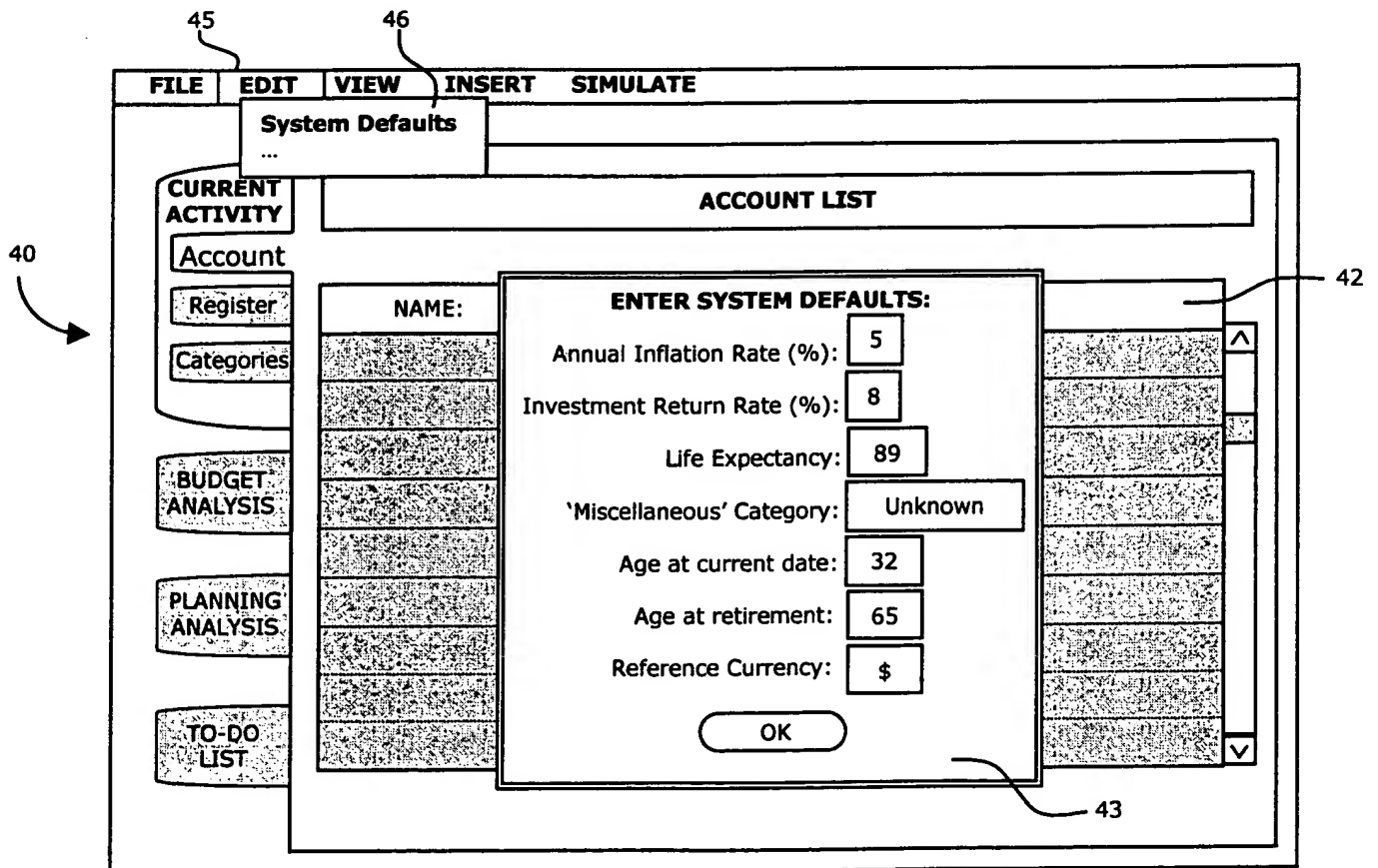


FIG. 4

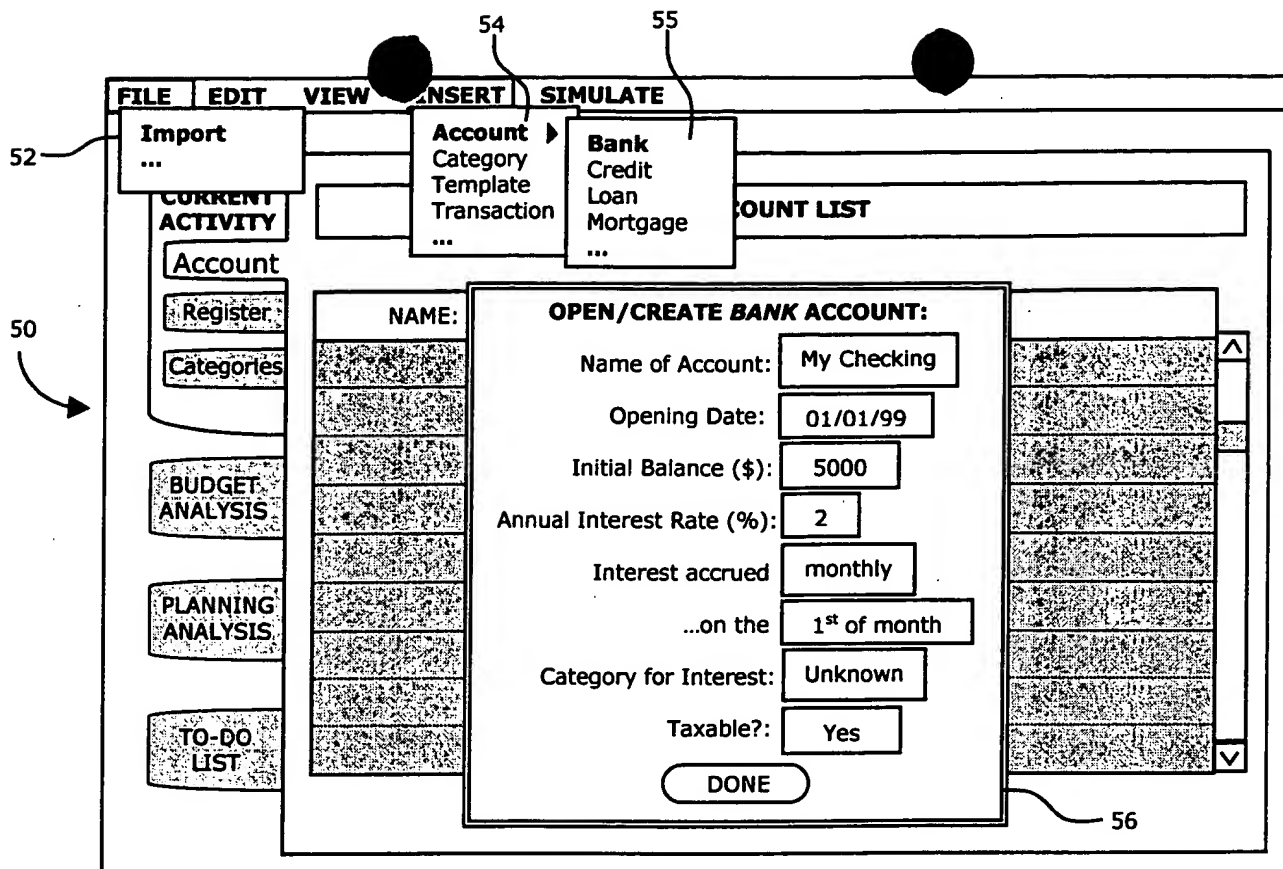


FIG. 5

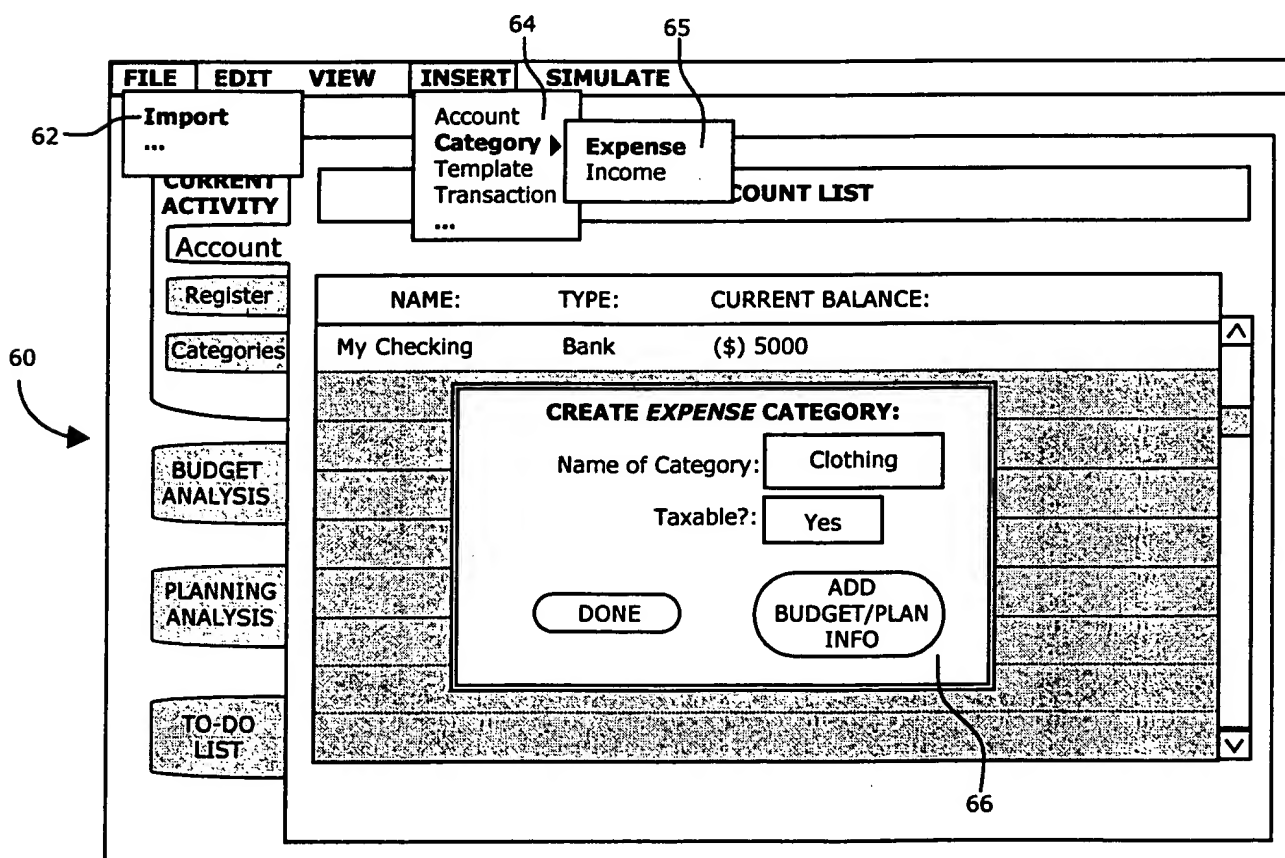


FIG. 6

FILE EDIT VIEW **INSERT** SIMULATE

Account  
Category  
Template  
Transaction  
...

Scheduled Spending Activity  
Scheduled Income Activity  
...

**CURRENT ACTIVITY**

Accounts

Register

Category

**BUDGET ANALYSIS**

**PLANNING ANALYSIS**

**TO-DO LIST**

**SPENDING ACTIVITY TEMPLATE:**

Spending Category is: Clothing

Description: Personal Spending on Clothing

Starting Date: 12/15/98 Ending Date: (End of Time)

Spend (\$) 200 on a monthly basis, from account: My Checking

Inflate Spending at an annual rate of (%): (Inflation = 5%)

Inflate on a yearly basis, on the 1<sup>st</sup> of January

DONE

FIG. 7

FILE EDIT VIEW **INSERT** SIMULATE

**PLANNING ANALYSIS**

All  
Template  
Object

**BUDGET ANALYSIS**

**CURRENT ACTIVITY**

**TO-DO LIST**

Dec 1998 Feb 1999 Apr 1999 Jun 1999 Aug 1999

START OF PLAN

Category: Clothing

Transaction: My Checking to Clothing

Account(Bank): My Checking

1998 2060

View

FIG. 8

BEST AVAILABLE COPY

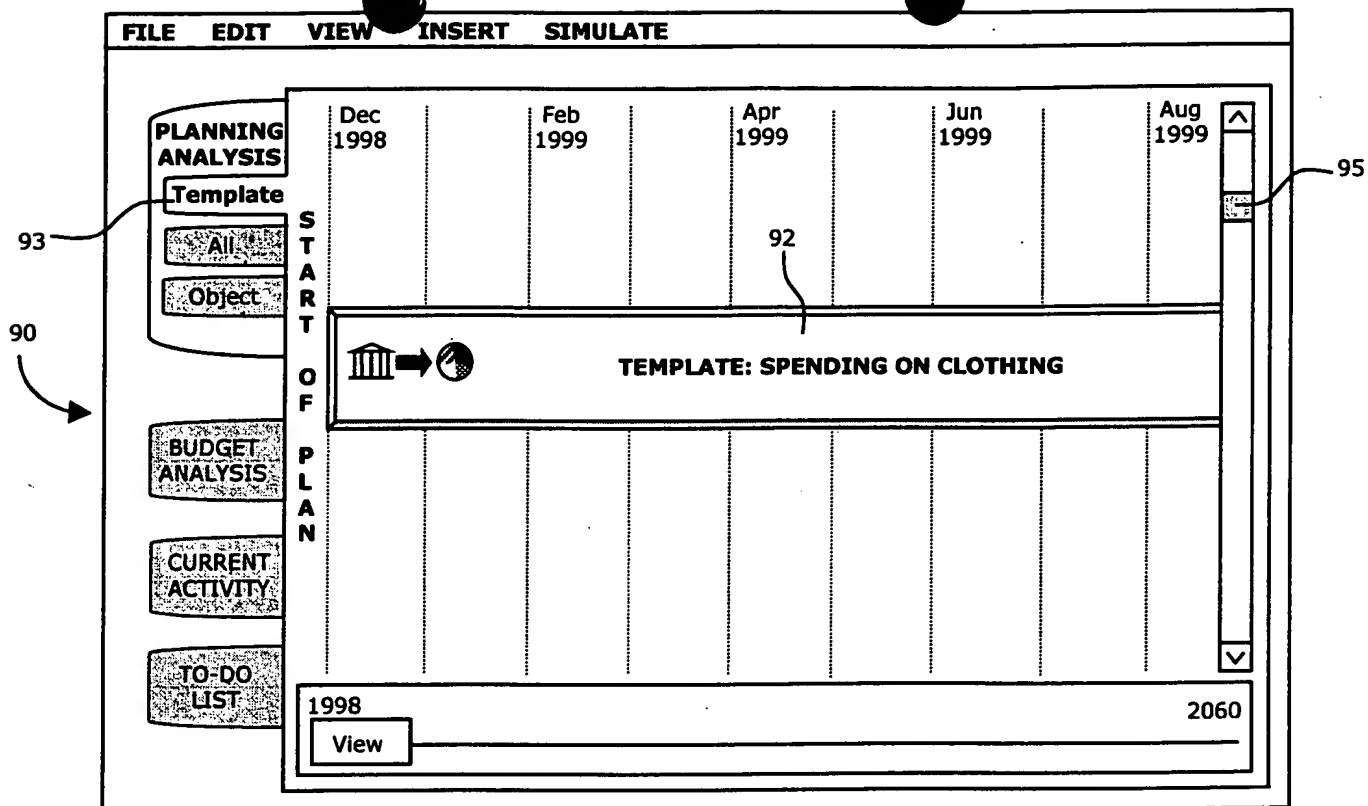


FIG. 9

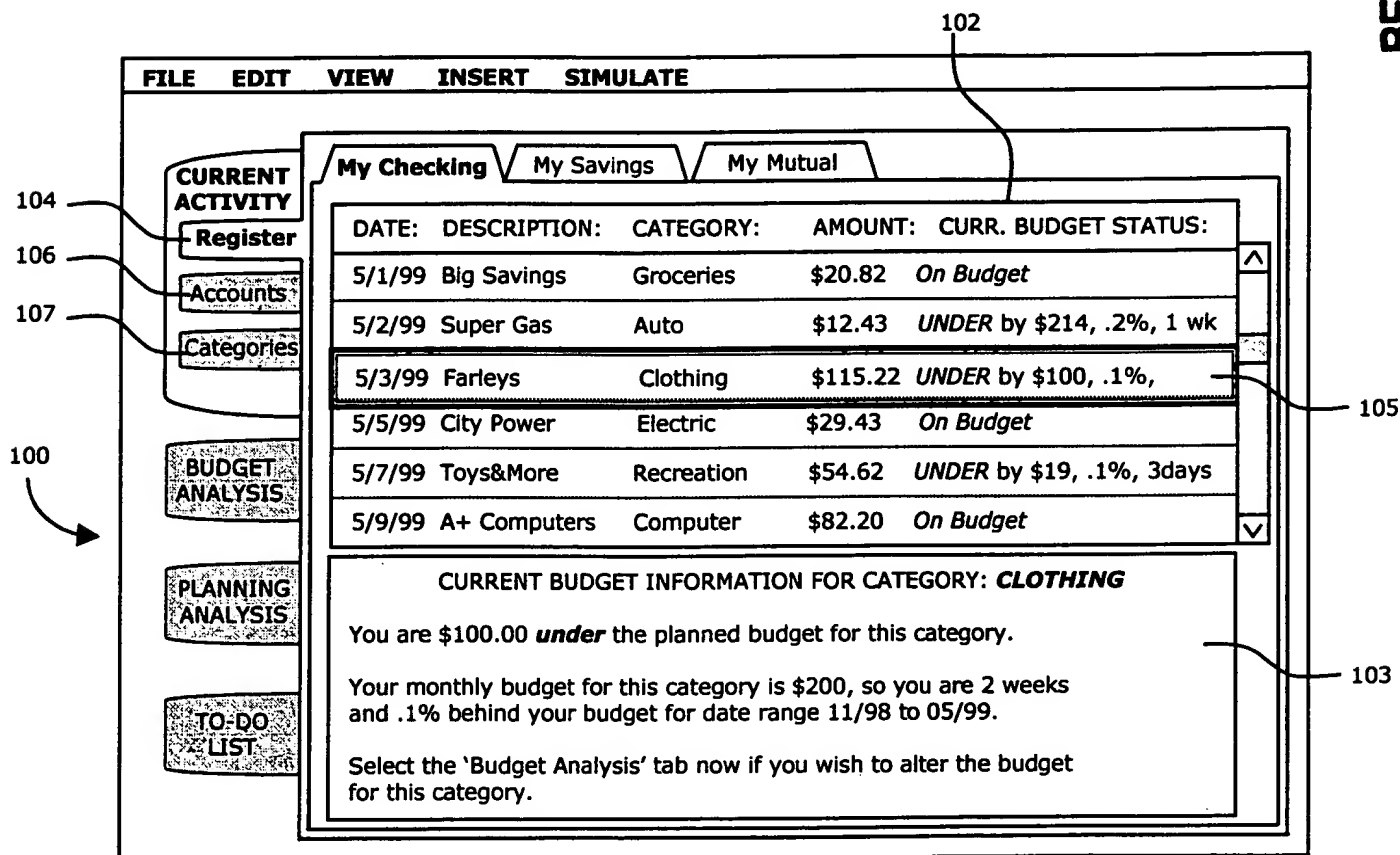
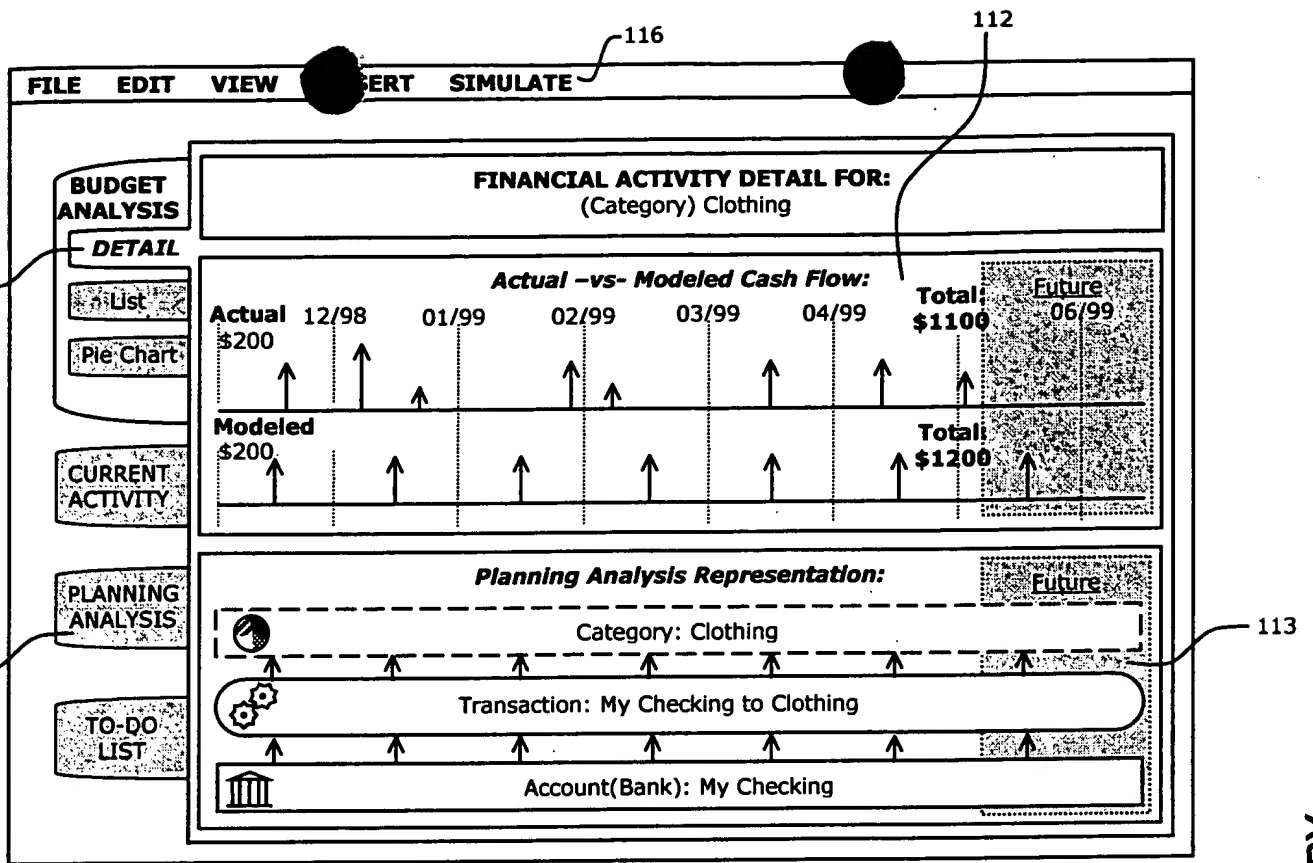
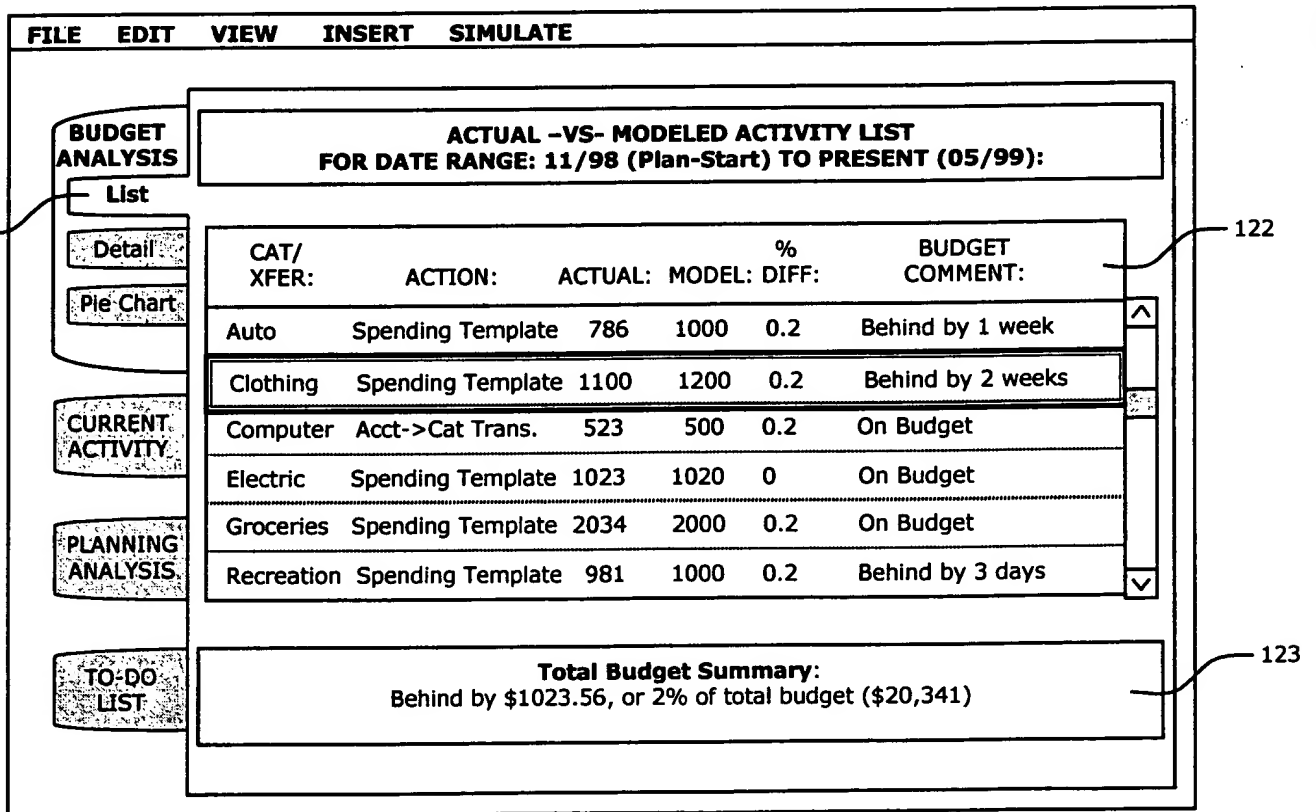


FIG. 10



**FIG. 11**



**FIG. 12**

FILE EDIT VIEW **INSERT** SIMULATE

Account Category  
**Template** Transaction ...

Scheduled Spending Activity  
 Scheduled Income Activity  
**Loan Account Payment Activity**  
 ...

Feb 2011

**PLANNING ANALYSIS**

All

Object

Template

**BUDGET ANALYSIS**

**CURRENT ACTIVITY**

**TO-DO LIST**

**ASSET LOAN ACCOUNT PAYMENT TEMPLATE:**

Description: My Auto #1

Starting Date: 7/1/2006 Ending Date: 7/30/2010

Pay out of Account: MyChecking Amounts are in 1999 values

Principle Category: Auto:Loan Interest Category: Int. Exp.

Asset Value (\$) 21000 Depreciation/Year(%): 5

Down Payment (\$) 10 % A.P.R. (%) : 6.9

DONE

1998 02/2010 02/2011 2060

View

FIG. 13

FILE EDIT VIEW **INSERT** SIMULATE

Feb 2010 May 2010 Aug 2010 Nov 2010 Feb 2011

**PLANNING ANALYSIS**

All

Object

Template

**BUDGET ANALYSIS**

**CURRENT ACTIVITY**

**TO-DO LIST**

Category: Clothing

Transaction: My Checking to Clothing

Account(Bank): My Checking

Trans: Loan

Account(Loan): My Auto #1

1998 02/2010 02/2011 2060

View

FIG. 14

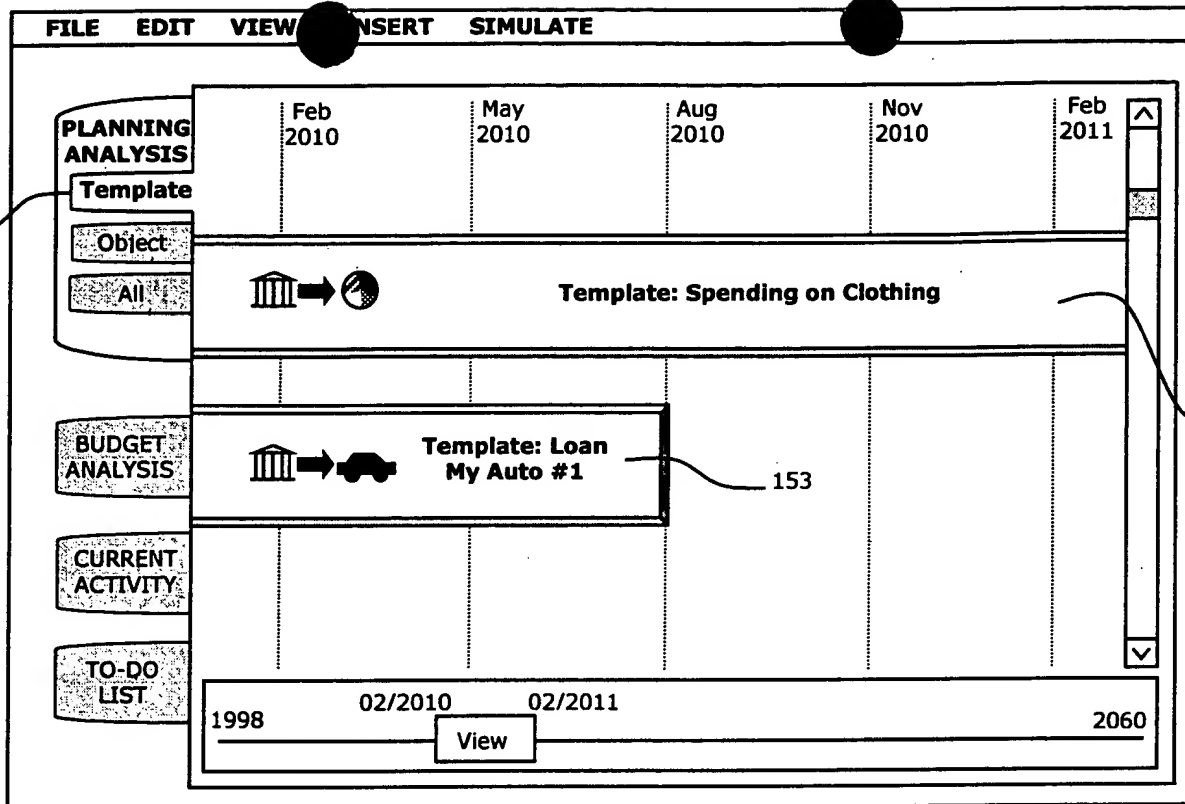


FIG. 15

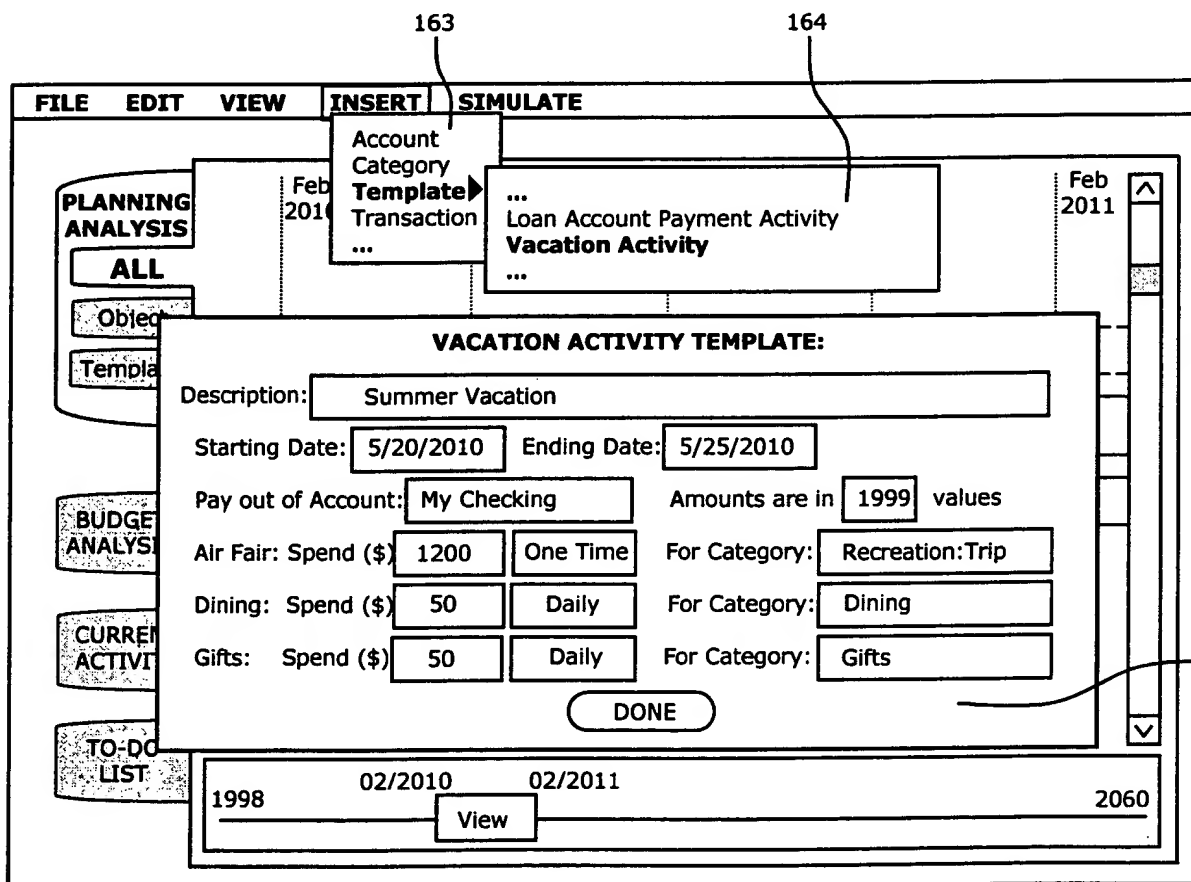


FIG. 16

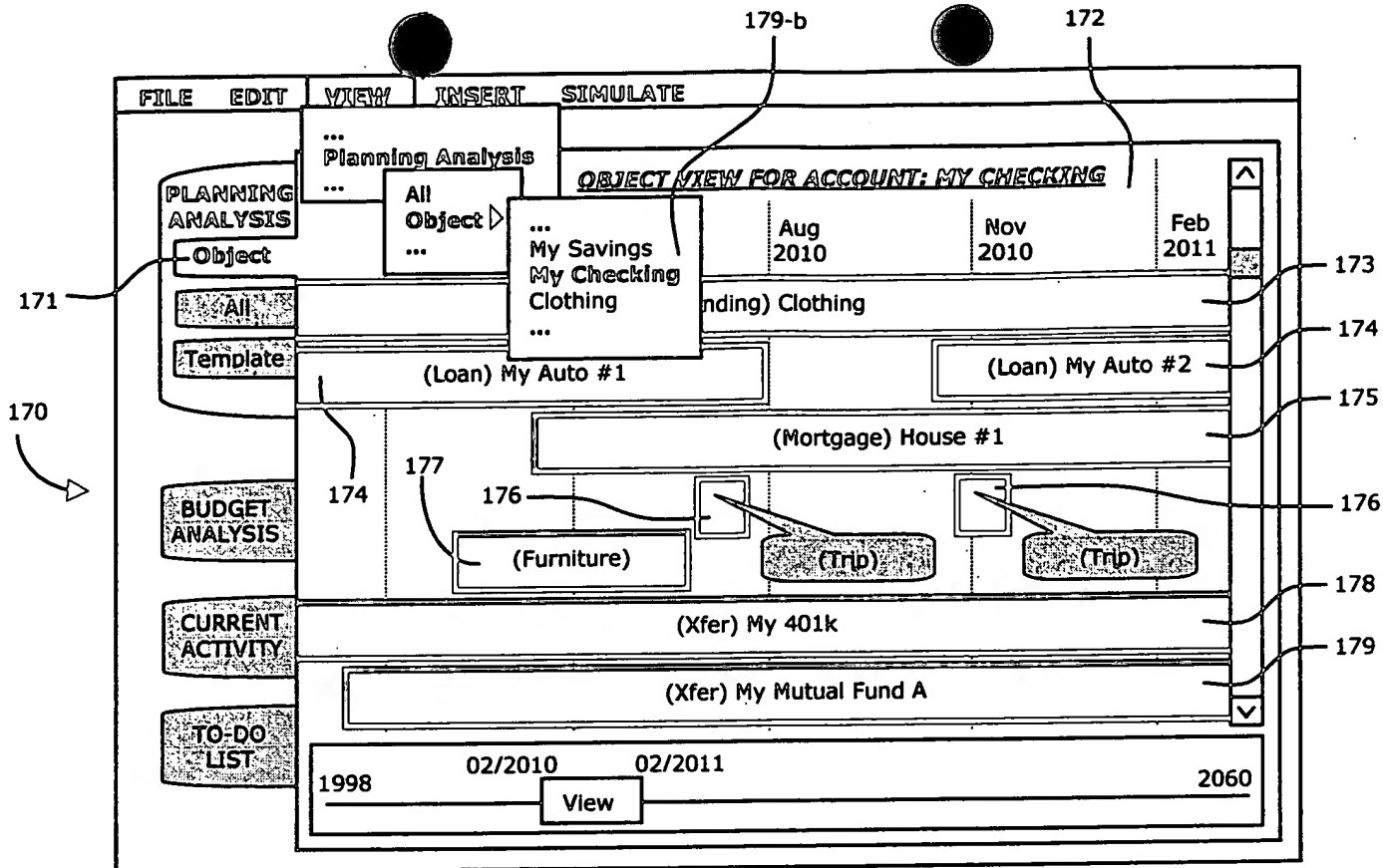


FIG. 17

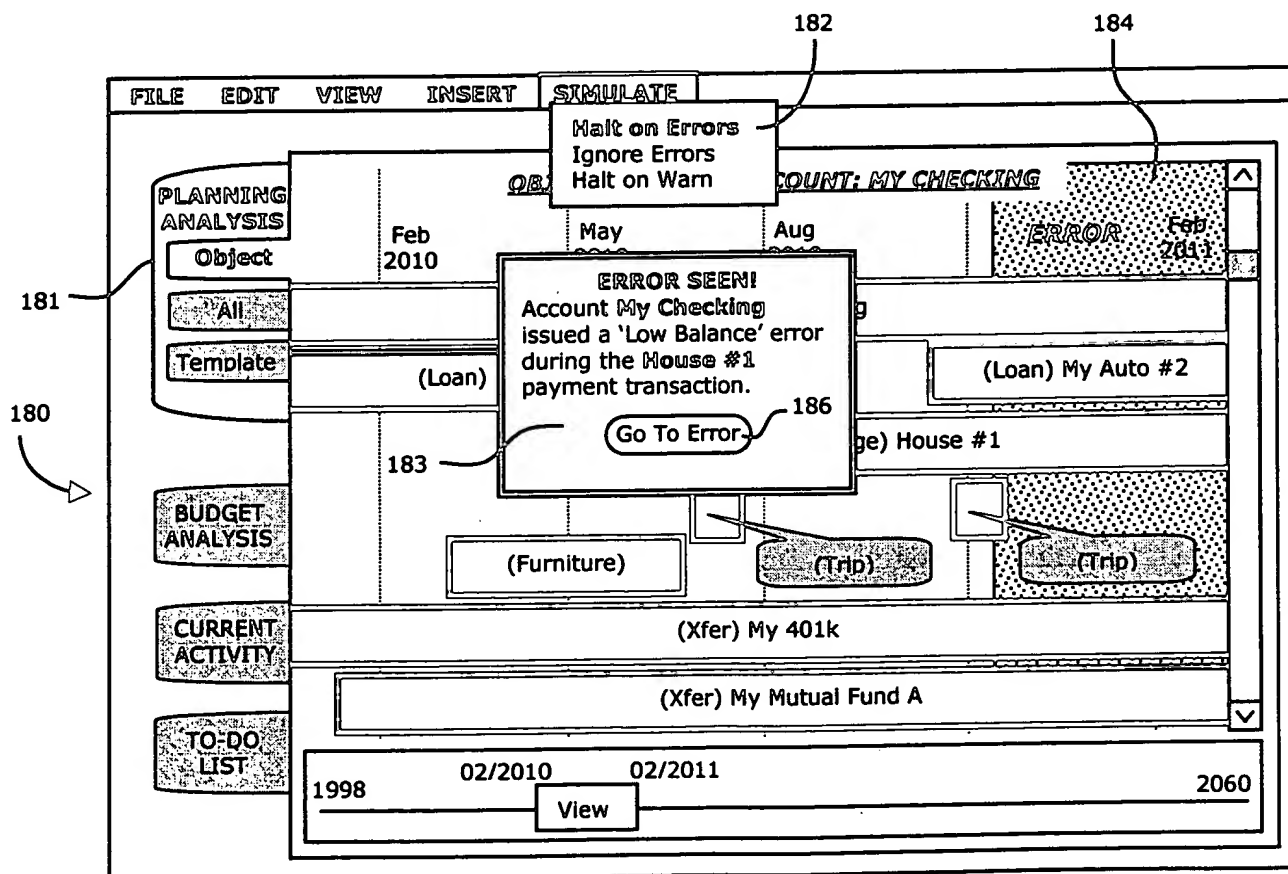


FIG. 18

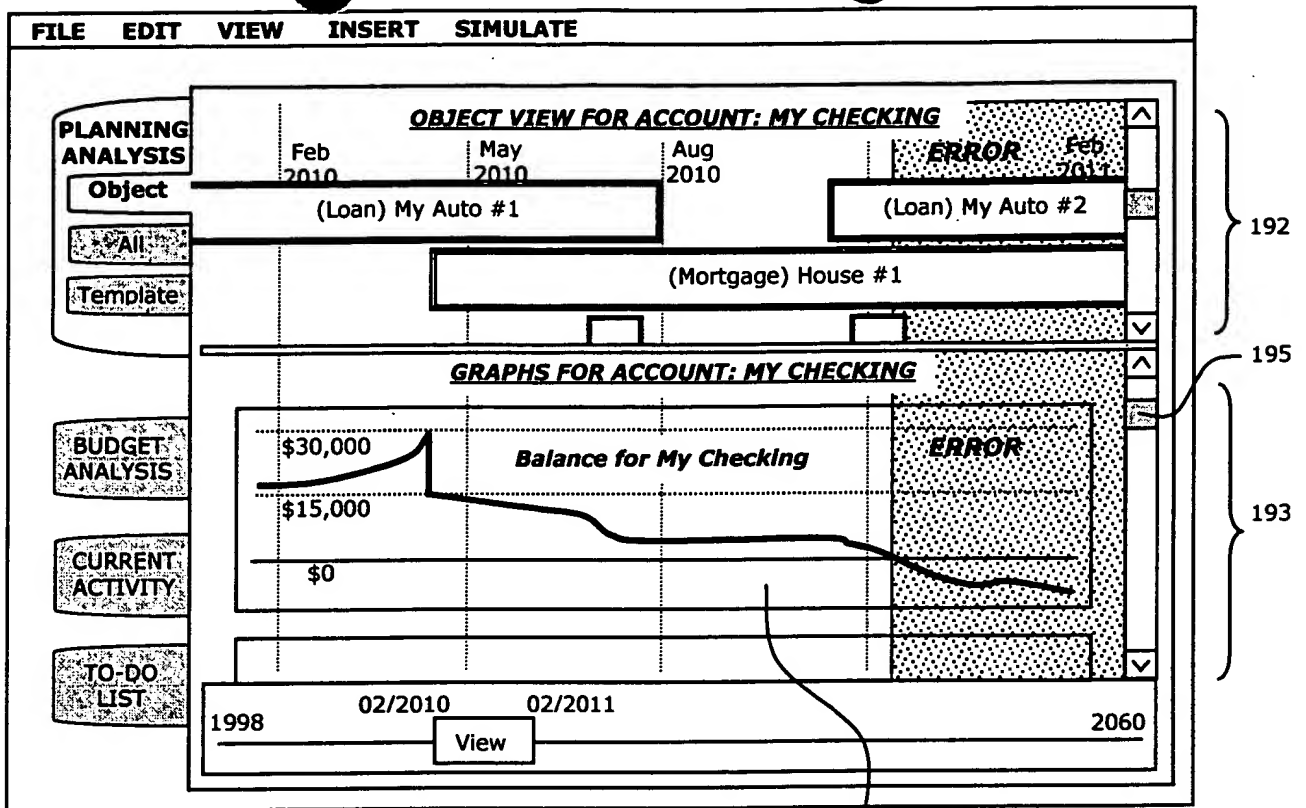


FIG. 19

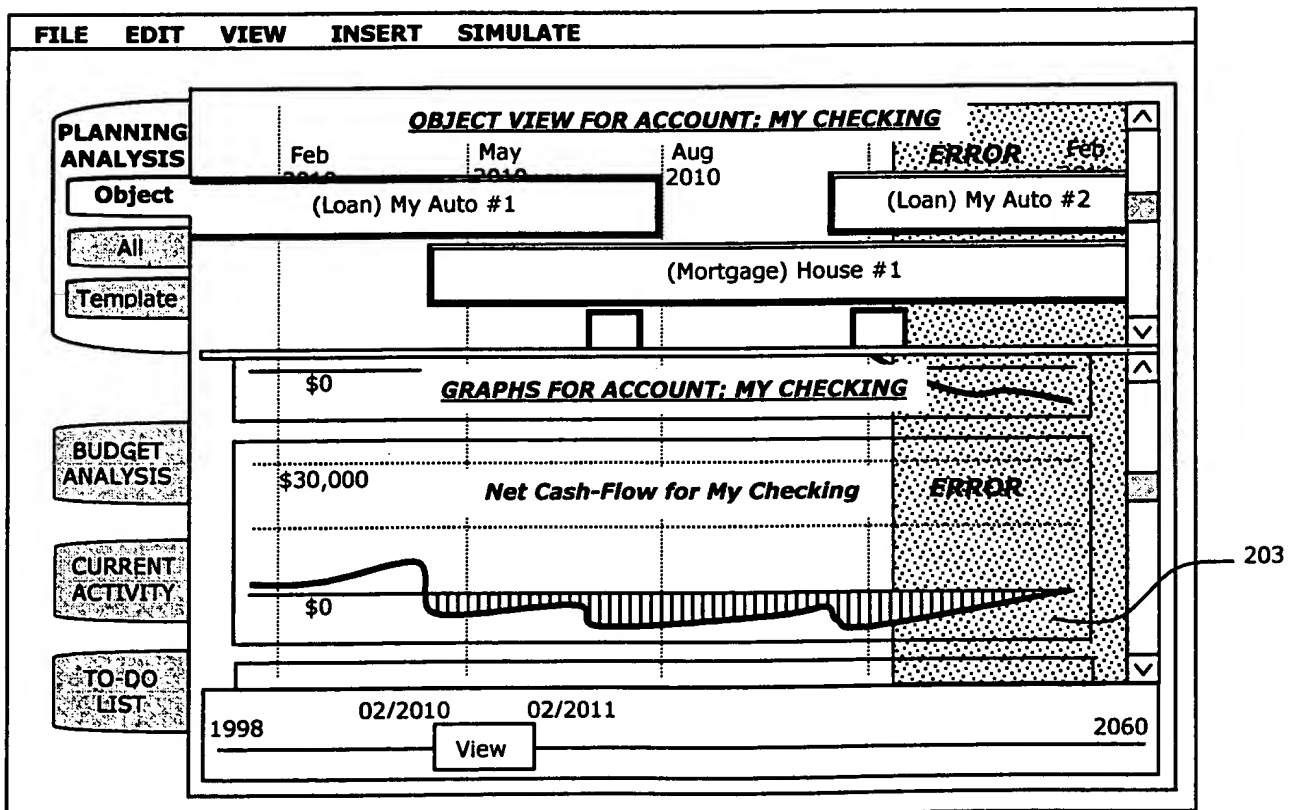


FIG. 20

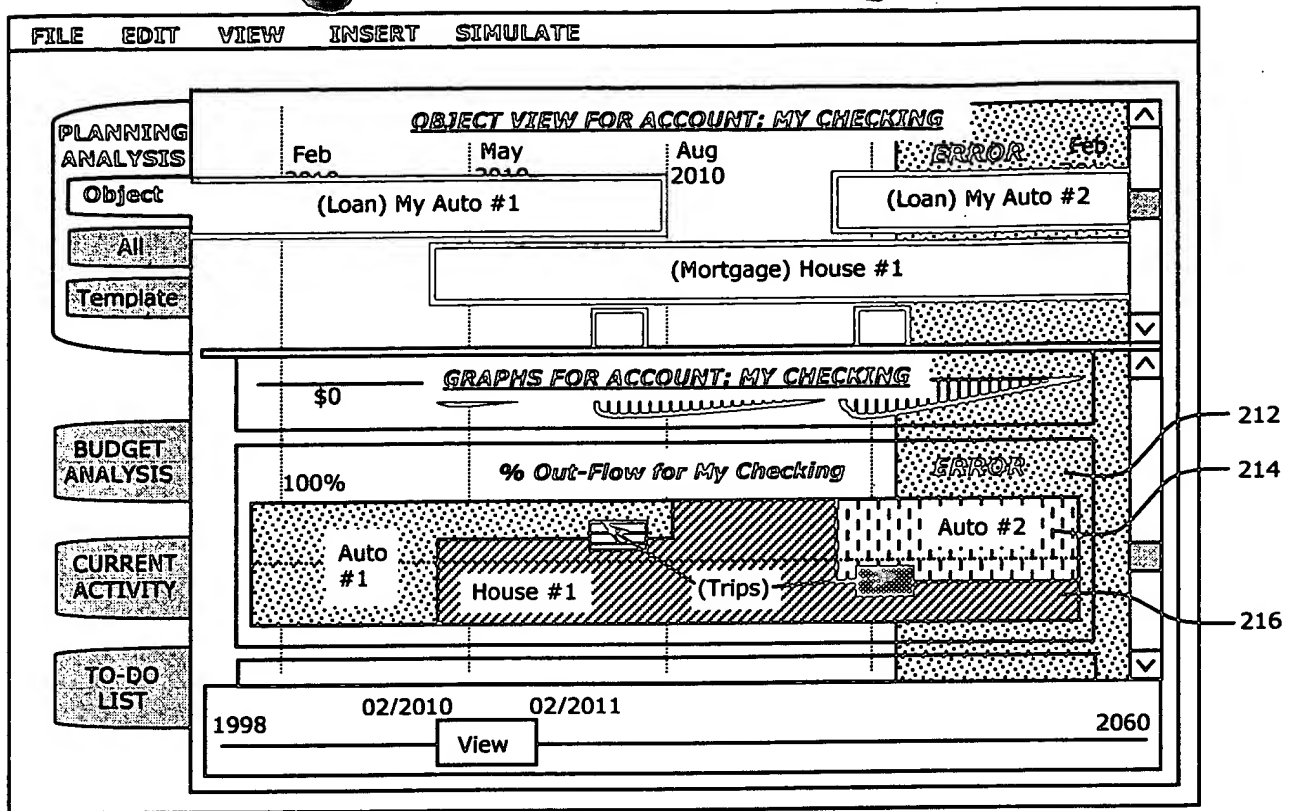


FIG. 21

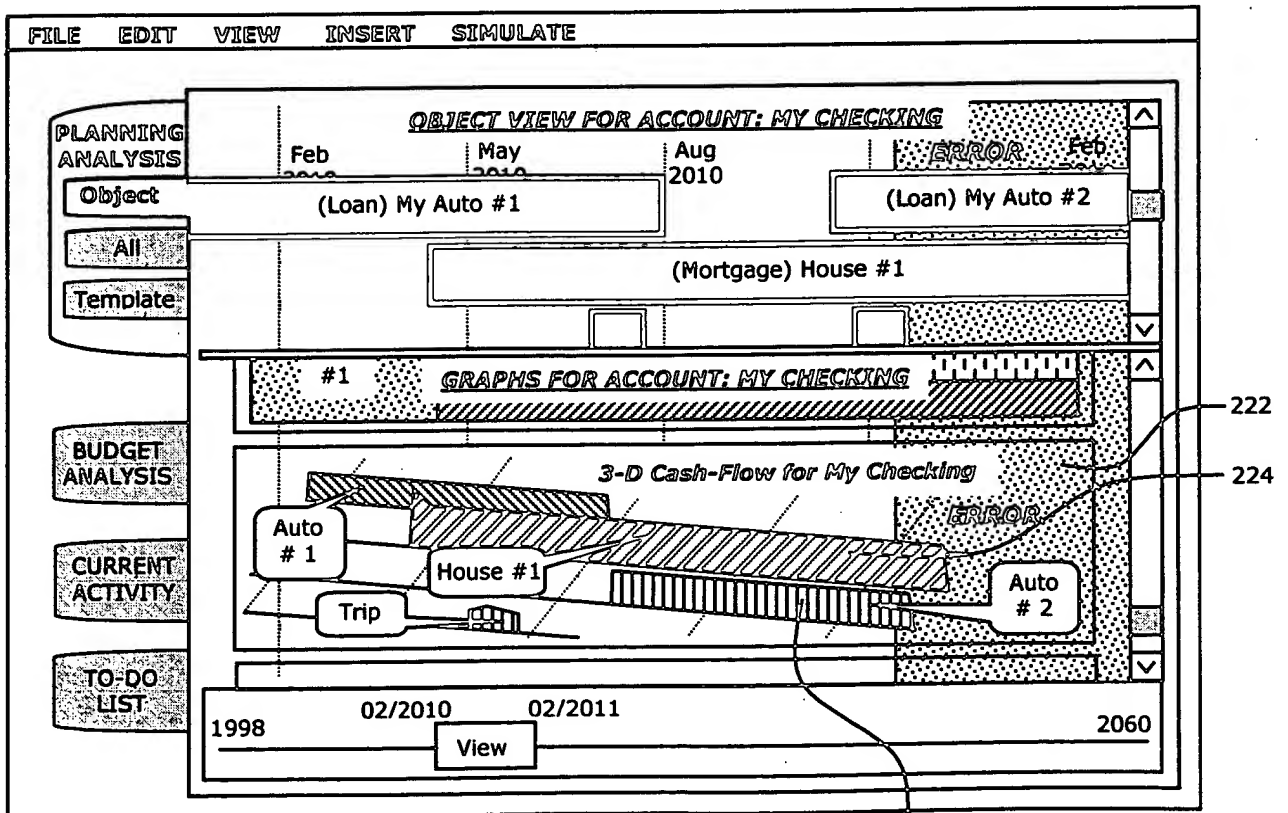


FIG. 22

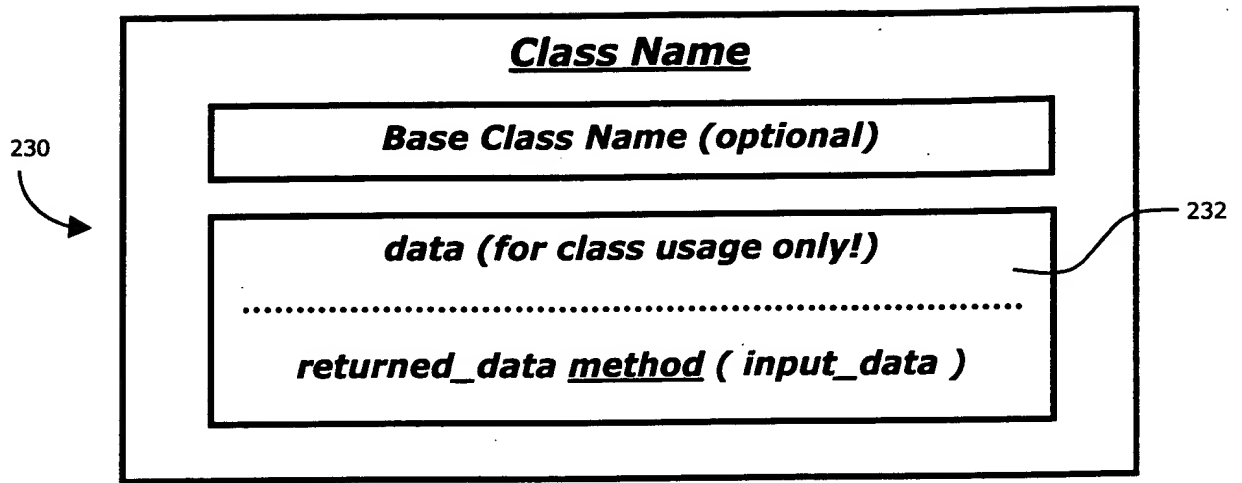


FIG. 23

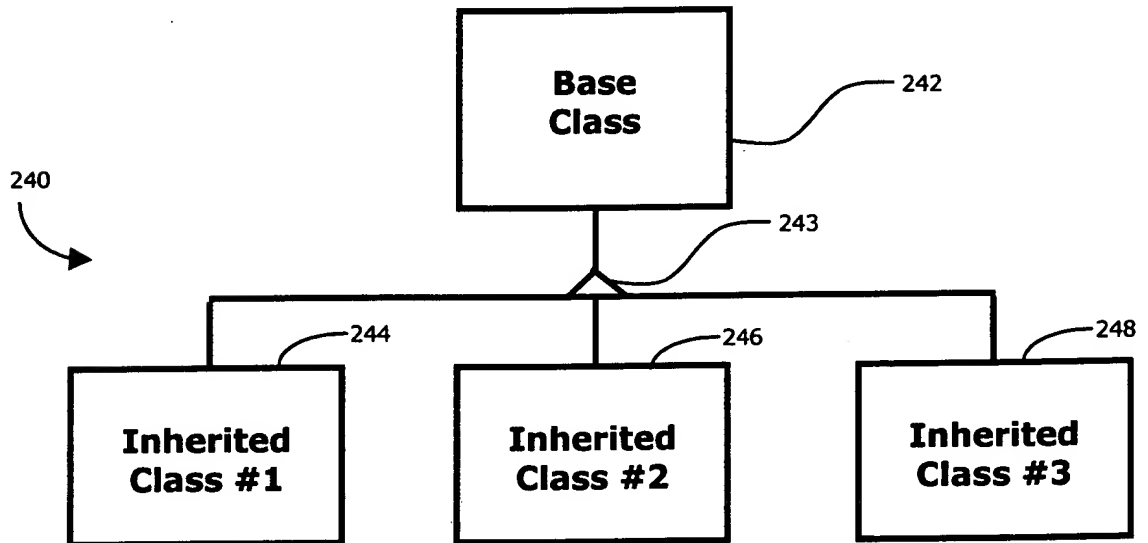


FIG. 24

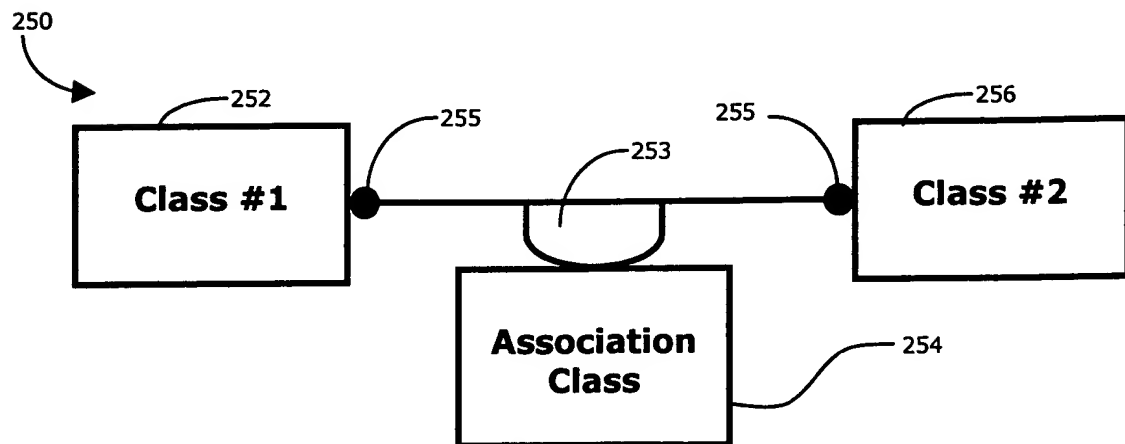
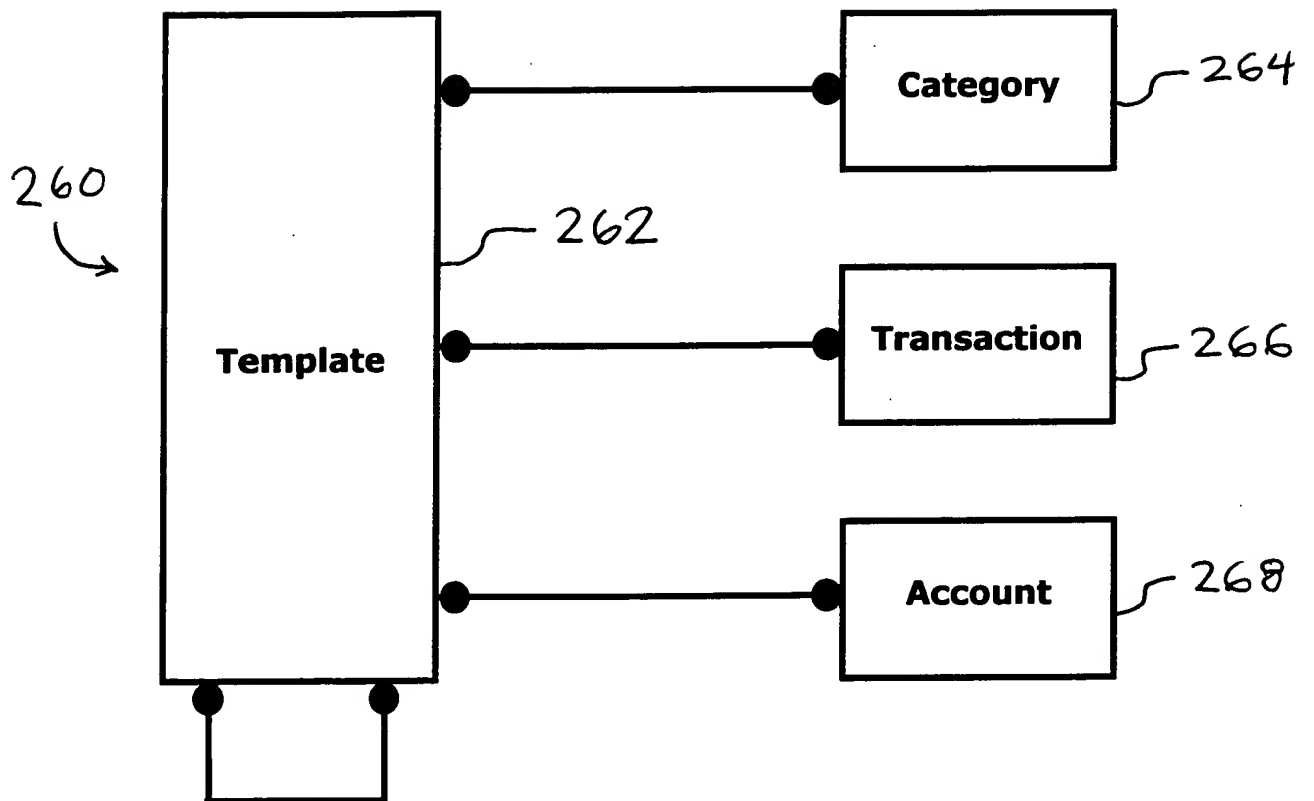


FIG. 25



**FIG. 26**

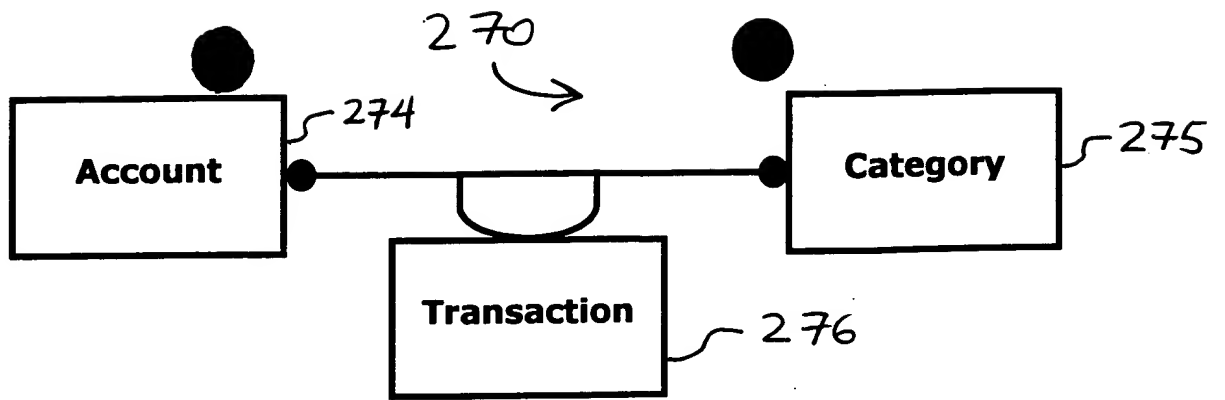


FIG. 27A

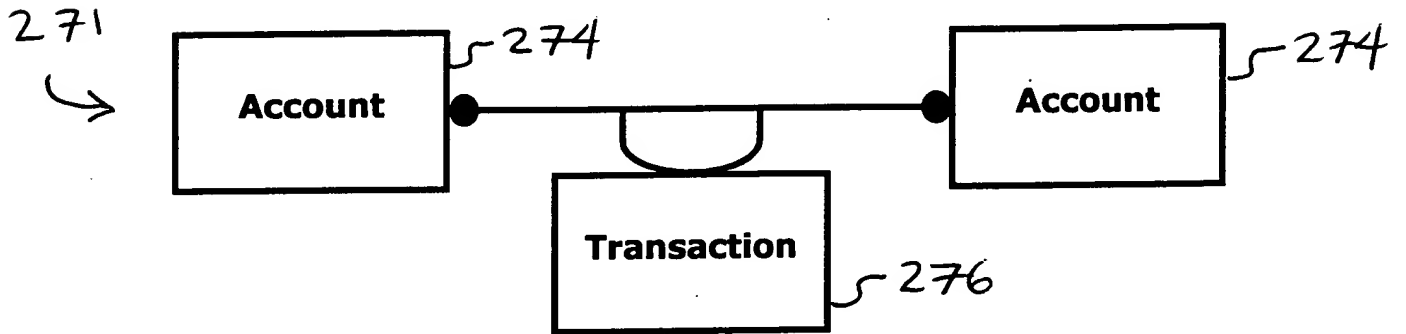


FIG. 27B

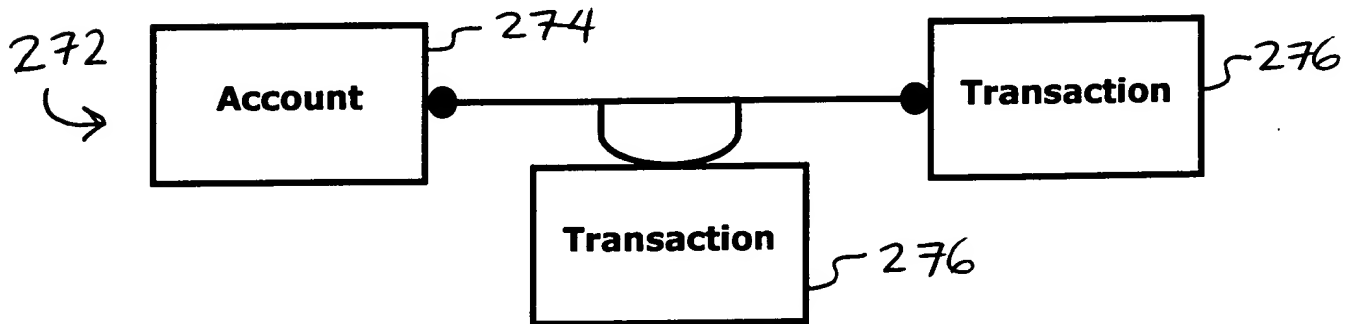


FIG. 27C

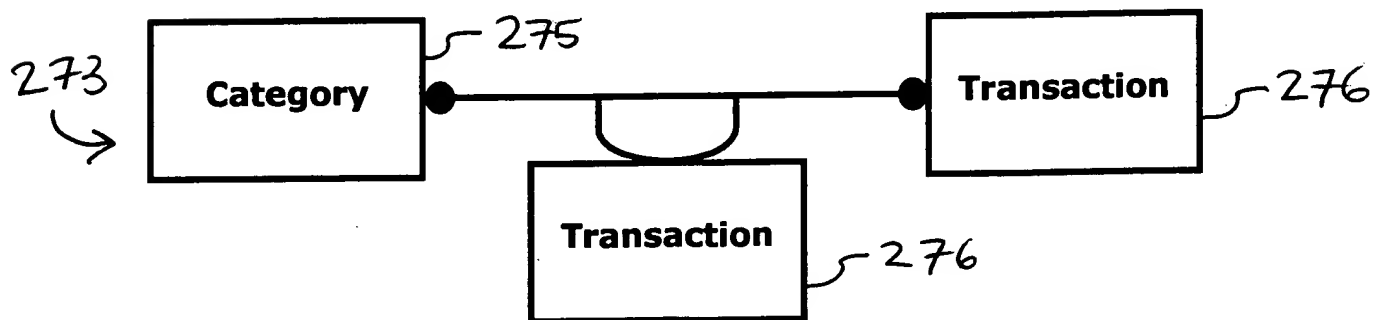


FIG. 27D

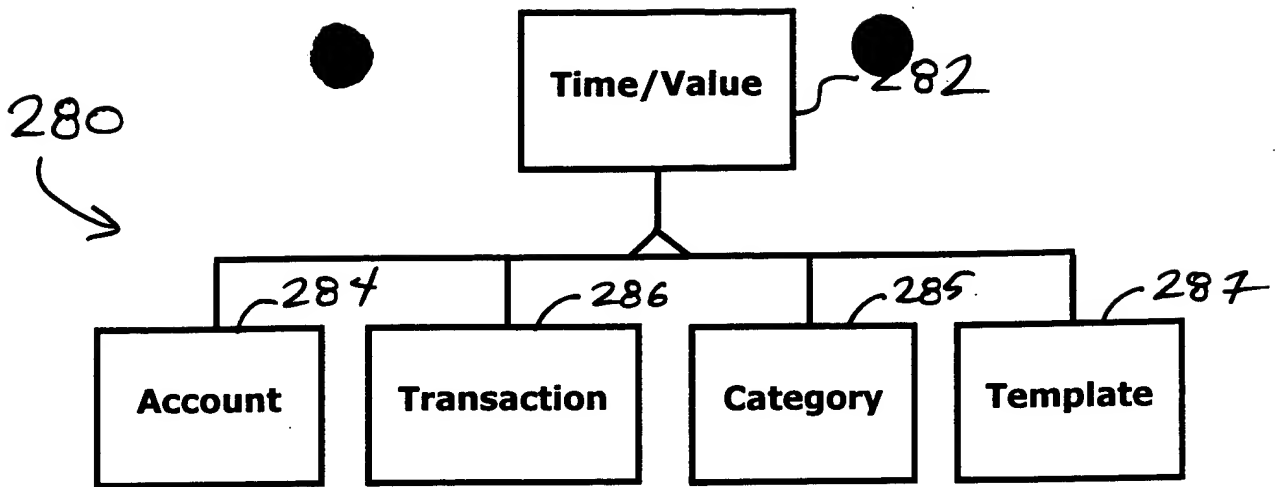


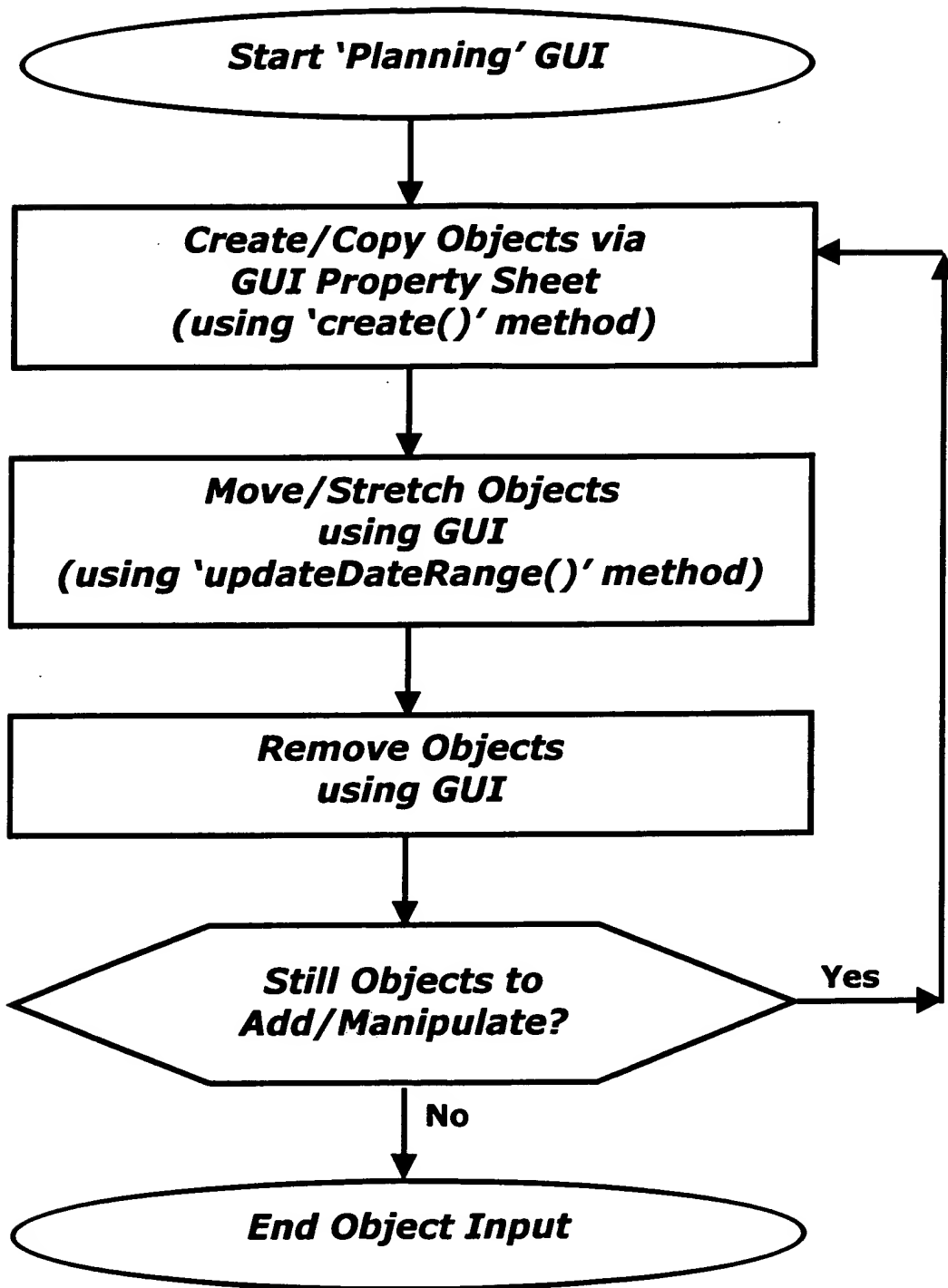
FIG. 28

**Time/Value (Base) Class**

User Description String  
 Object Reference String  
 Time/Value Linked List  
 State (Idle, Active)  
 Tax Information  
 Graphical Data (position, web-links, style, etc.)  
 Template Reference List (optional)  
 Notification Reference List (optional)  
 .....  
create( start\_date, start\_value,  
 stop\_date, ... )  
 date\_first resetToFirstDate()  
updateDateRange( start\_date\_new,  
 stop\_date\_new )  
addObjectRef( ref )  
removeObjectRef( ref )  
addNew( date, value )  
 value getValue( date )  
 event notifyReceive()

FIG. 29

300  
↪



**FIG. 30**

## **Account Class**

**Time/Value (Base) Class**

**Minimum/Maximum Limits**

**Current Activity Tool Object Reference List**

**create( name, type, opening\_date,**

**stop\_date, ... )**

**value getBalance()**

**value getWarningBalance()**

**value getErrorBalance()**

**open( cash\_ref )**

**close( cash\_ref )**

**deposit( cash\_ref )**

**withdraw( value, cash\_ref )**

**FIG. 31**

## **Category Class**

**Time/Value (Base) Class**

**Category Type (expense, income)**

**create( name, type, ... )**

**addExpense( cash\_ref )**

**getIncome( value, cash\_ref )**

**FIG. 32**

```
classDiagram
    class TransactionClass {
        <<abstract>>
        +Time/Value (Base) Class
        +Scheduling Information Object (update)
        +Scheduling Information Object (adjust)
        +Priority (0=lowest)
        +create( ... )
        +date_next updateWithDate( date_curr )
    }
    class TransactionClassSub {
    }
    TransactionClass <|-- TransactionClassSub
```

The diagram shows a UML class hierarchy. At the top is the **Transaction Class**, which is an abstract class. It has several attributes: **Time/Value (Base) Class**, **Scheduling Information Object (update)**, **Scheduling Information Object (adjust)**, **Priority (0=lowest)**, a **create( ... )** method, and a **date\_next updateWithDate( date\_curr )** method. Below it is a concrete class, **Transaction Class**, which inherits from the abstract class.

### ***Time/Value (Base) Class***

### Scheduling Information Object (update)

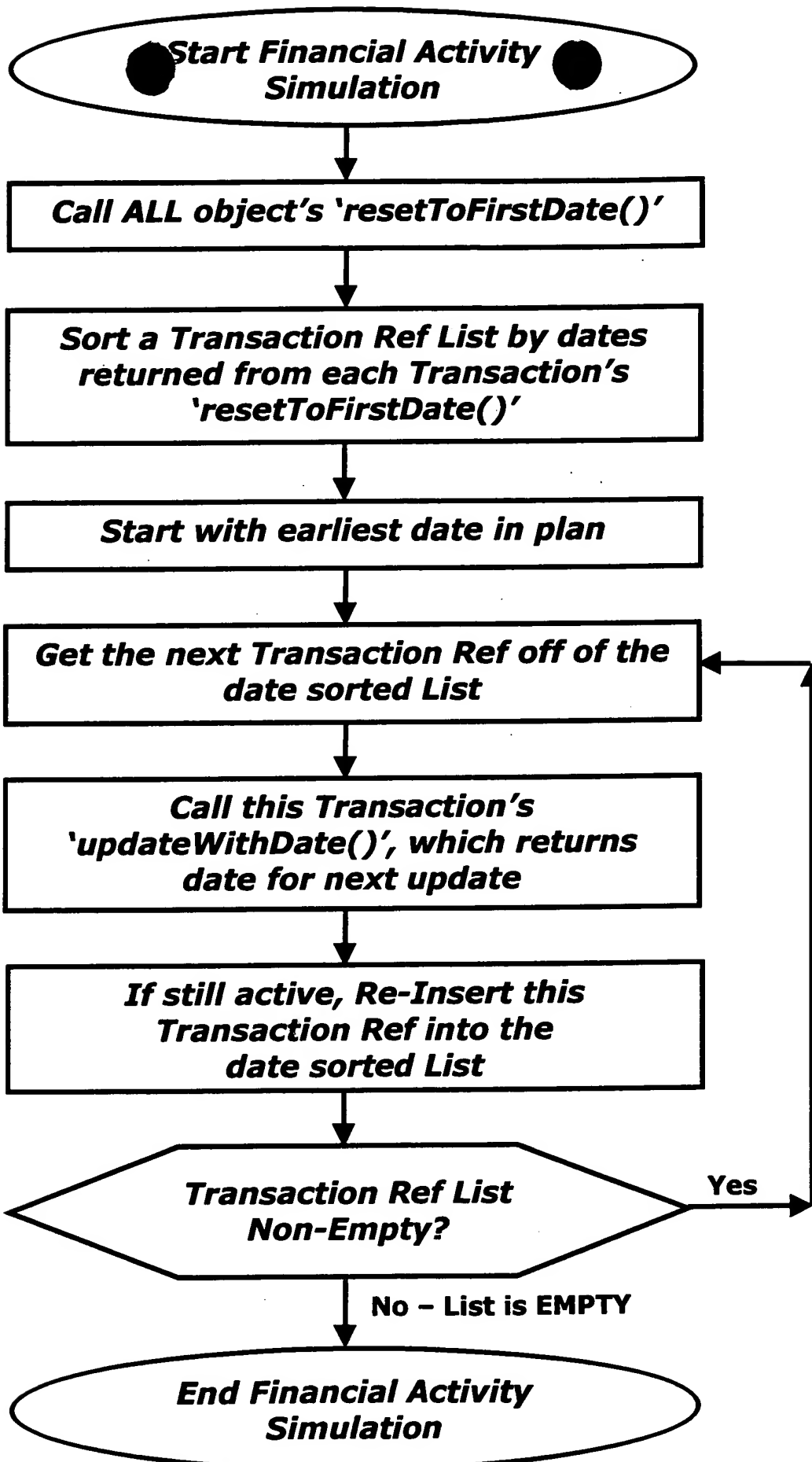
### Scheduling Information Object (adjust)

**Priority (0=lowest)**

```
.....
create( ... )
.....
```

```
date_next updateWithDate( date_curr )
```

**FIG. 33**



BEST AVAILABLE COPY

FIG. 34

### **System Interface Class**

**Inflation-Rate-%/Year Linked List**

**Market-Return-%/Year Linked List**

**Current Age, Retirement, Life Expectancy**

**'Miscellaneous' Category Reference**

**Reference Currency (\$ or foreign)**

.....  
**create( ... )**

**date getCurrentDate()**

**value getInflationPct( date )**

**value getMarketReturnPct( date )**

**value getInflatedValue( value\_from,**

**date\_from, date\_to )**

**throwWarning( code )**

**throwError( code )**

**print( format\_string, ... )**

**createCash( value, cash\_ref )**

**returnValue( value, string\_ref )**

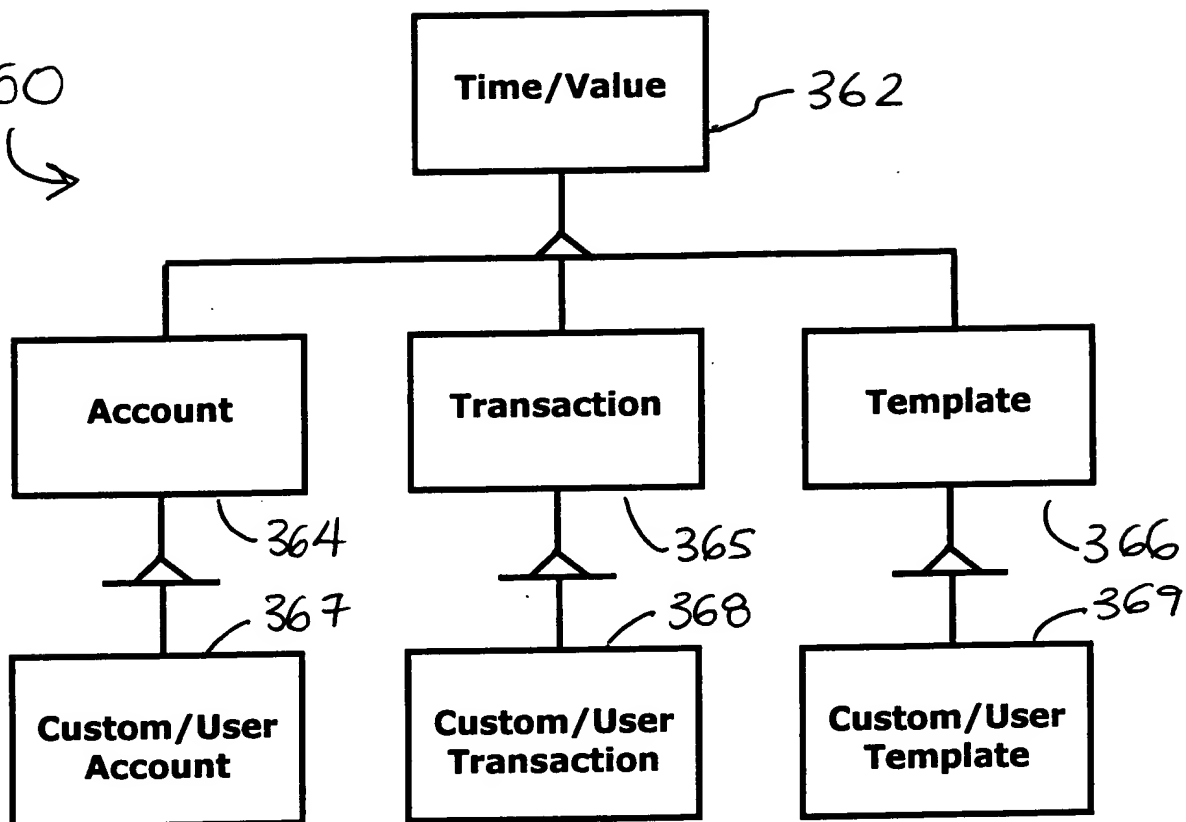
**returnCash( cash\_ref, string\_ref )**

**notifySend( target\_object\_reference, event )**

**notifyAll( event )**

**FIG. 35**

360  
→



**FIG. 36**

**Scheduling Information Class**

**First Date**

**Last Date**

**Next Scheduled Date**

**Scheduling Method (daily, weekly, monthly, etc.)**

**Scheduling Frequency (every time,  
every other time, every 3<sup>rd</sup>, etc.)**

.....  
**create( ... )**

**date resetToFirstDate()**

**date getNextDate()**

**setNextDate( date )**

**date computeNextDate()**

**updateDateRange( start\_date\_new,  
stop\_date\_new )**

**FIG. 37**

**Account-to-Account Transfer  
Transaction Class**

**Transaction (Base) Class**

**'From' Account Object Reference**

**'To' Account #2 Object Reference**

**Transfer Amount value**

**Adjustment Percentage**

.....  
**create( ... )**

**FIG. 38**

FILE EDIT VIEW INSERT SIMULATE

Account Category Template Transaction > Account To Account Transfer ...

Feb 2010 Aug Nov 2010 Feb 2011

PLANNING ANALYSIS

All

Object

Temp

**ACCOUNT TO ACCOUNT TRANSFER TRANSACTION:**

Description: My Checking transfer to My Savings

Starting Date: 7/1/2006 Ending Date: 7/30/2010

Withdraw From Account: My Checking Deposit To Account: My Savings

Transfer Amt (\$) (Enter value, or hit F1) every month

Adjust Pct (%) (Enter value, or hit F1) every year

DONE

1998 02/2010 02/2011 2060

View

FIG. 39

FILE EDIT VIEW INSERT SIMULATE

Feb 2010 May 2010 Aug 2010 Nov 2010 Feb 2011

PLANNING ANALYSIS

All

Object

Temp

**ACCOUNT TO ACCOUNT TRANSFER TRANSACTION:**

Description: My Checking transfer to My Savings

Starting Date: 7/1/2006 Ending Date: 7/30/2010

Withdraw From Account: My Checking Deposit To Account: My Savings

Transfer Amt (\$) Select one option from list For 'Transfer Amt (\$)' value: every month

Adjust Pct (%) System methods > getInflation() getInflatedValue() > every

Returned values

GetInflatedValue: Value From (\$) 150 Date From 1999 Date To: (Current)

1998 02/2010 02/2011 2060

View

FIG. 40

**FILE EDIT VIEW INSERT SIMULATE**

**PLANNING ANALYSIS**

**All**

**Object**

**Temp**

**BUDG ANALY**

**CURR ACTIV**

**TO-DO LIST**

Feb 2010 May 2010 Aug 2010 Nov 2010 Feb 2011

**ACCOUNT TO ACCOUNT TRANSFER TRANSACTION:**

Description: My Checking transfer to My Savings

Starting Date: 7/1/2006 Ending Date: 7/30/2010

Withdraw From Account: My Checking Deposit To Account: My Savings

Transfer Amt (\$) 150.00 (in 1999 cash) every month

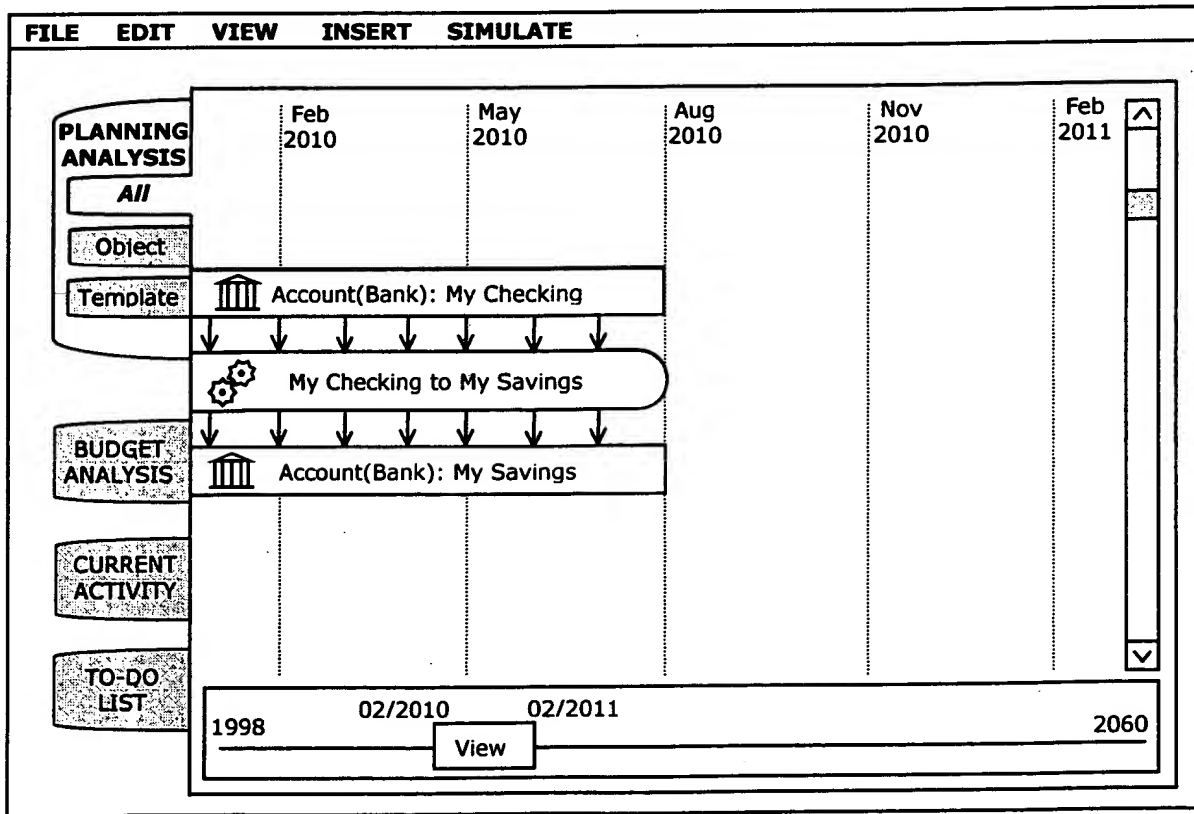
Adjust Pct (%) (User Inflation Rate) every year

**DONE**

1998 02/2010 02/2011 2060

**View**

**FIG. 41**



**FIG. 42**

### **Transaction Class**

430  
→

```
class CTrans : public CTimeValue           // A pure-virtual/abstract base class!
{
public:
    CTrans(
        const char    *name,                // Transaction's reference-name
        ...              // More input parameters (not shown)
    );
    virtual ~CTrans();

    virtual CDate updateWithDate( CDate date_curr ) = 0; // PURE VIRTUAL!
                                                    // ...Must inherit this class!

    Virtual CDate resetToFirstDate();           // Inherited from Time/Value
    Virtual void updateDateRange( CDate date_start, CDate date_stop );
                                                    // Inherited from Time/Value

protected:
    CScheduler        m_schUpdate;           // Schedules next update date
    CScheduler        m_schAdjust;           // Schedules next adjust date
    priority_t        m_priority;           // Priority (0=lowest)
}; // END of 'CTrans' class
```

**FIG. 43**

### **Account-to-Account Transfer Transaction Class**

440  
→

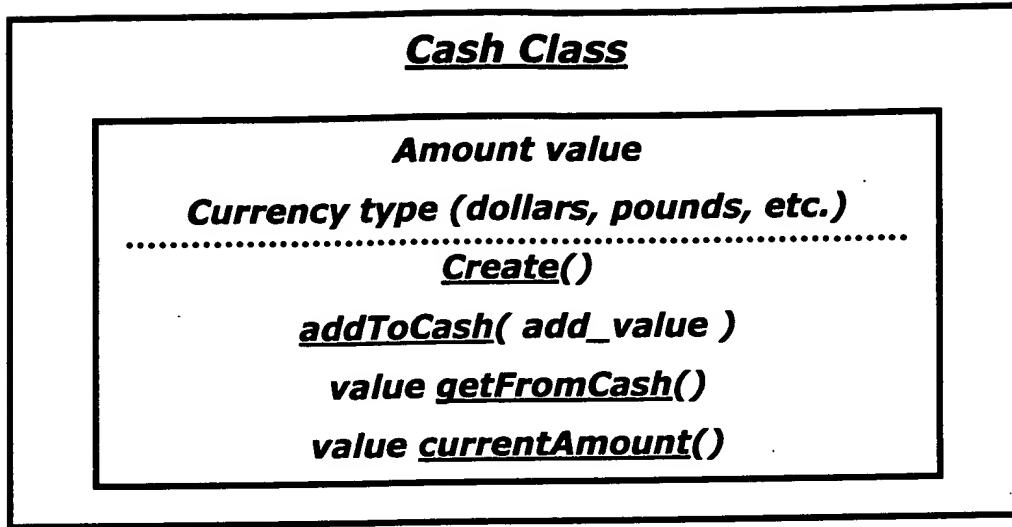
```
class CTrans_acctToAcct : public CTrans
{
public:
    CTrans_acctToAcct(
        const char    *name,                // Transaction's reference-name
        ...              // More input parameters (not shown)
    );
    virtual ~CTrans_acctToAcct();

    virtual CDate updateWithDate( CDate date_curr );

    Virtual CDate resetToFirstDate();
    Virtual void updateDateRange( CDate date_start, CDate date_stop );

protected:
    CAccount          *m_acctFrom;           // Xfer 'From' this acct
    CAccount          *m_acctTo;            // Xfer 'To' this acct
    value_t           m_moneyToXfer;        // Money to transfer at each update
    value_t           m_adjustPct;          // % to adjust xfer amount
}; // END of 'CTrans_acctToAcct' class
```

**FIG. 44**



**FIG. 45**

## **Account-to-Account Transfer** **Transaction Class Method Example**

460 →

```
CDate CTrans_accntToAccnt::updateWithDate( CDate date_curr )
{
    Cdate date_test = SYSINTF.getCurrentDate(); // Not used here, just for demo
    if ( date_test == date_curr )
    {
        SYSINTF.print( "Just a test...dates are equal!" );
    }

    // Check to see if the simulated current date does NOT match our expected
    // current date, leaving if it doesn't (an invalid condition)
    if ( date_curr != m_schUpdate.getNextDate() )
    {
        SYSINTF.throwError( ERR_UNEXPECTED_DATE );
        return( date_curr ); // Terminates simulation for this transaction!
    }

    // Adjust parameters if the current simulation date matches or exceeds
    // our next adjustment date
    if ( date_curr >= m_schAdjust.getNextDate() )
    {
        m_moneyToXfer *= 1.0 + m_adjustPct / 100.0;
        m_schAdjust.computeNextDate(); // Set the next adjustment date
    }

    // CREATE a 'cash' data type (sets simulated cash amount to ZERO)
    CCash cash_xfer;

    // WITHDRAW cash FROM account (makes simulate cash a positive amount)
    m_acctFrom->withdraw( m_moneyToXfer, cash_xfer );

    // DEPOSIT cash TO account (makes simulated cash zero again, after transfer)
    m_acctTo->deposit( cash_xfer );

    // LOG this transfer amount to the Time/Value (base) class
    addNew( date_curr, m_moneyToXfer );

    // Return the date that we wish the Cash-Flow Simulator to call us with again
    return( m_schUpdate.computeNextDate() );

    // NOTE: When this method call returns, 'cash_xfer' will be AUTOMATICALLY
    // destroyed, which calls the 'cash' class' destructor method call. A NON-ZERO
    // simulated cash amount in 'cash_xfer' would cause a system warning!
} // END of 'CTrans_accntToAccnt::updatePerDate()'
```

**FIG. 46**